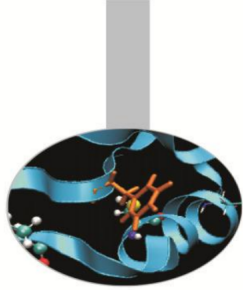
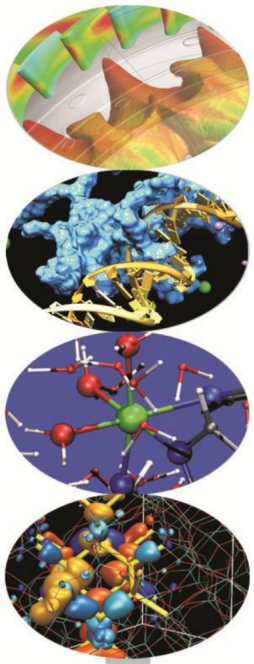
CINECA SCAI
SuperComputing Applications and Innovation



TensorFlow basics

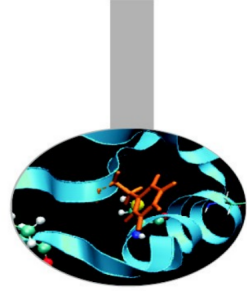


Riccardo Zanella – r.zanella@cineca.it

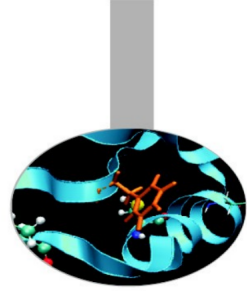
SuperComputing Applications and Innovation Department

CINECA

Table of Contents

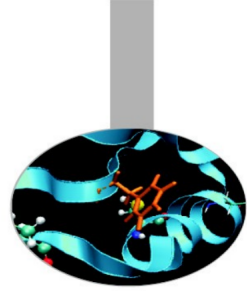


TensorFlow: basic usage



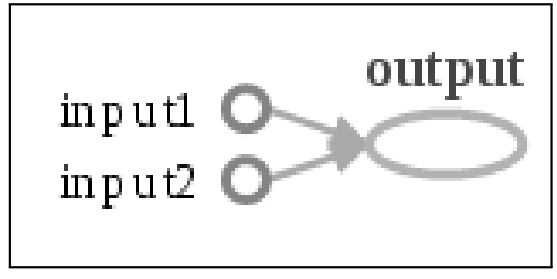
```
import tensorflow as tf

# define graph nodes (on default graph)
input1 = tf.constant([1.0, 1.0, 1.0, 1.0])
input2 = tf.constant([2.0, 2.0, 2.0, 2.0])
output = tf.add(input1, input2)

# launch a session
sess = tf.Session()

# run graph on session and retrieve results
result = sess.run(output)

# close the session
sess.close()
```

TensorFlow: basic usage

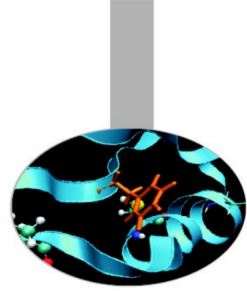


```
import tensorflow as tf

# define graph nodes (on default graph)
input1 = tf.constant([1.0, 1.0, 1.0, 1.0])
input2 = tf.constant([2.0, 2.0, 2.0, 2.0])
output = tf.add(input1, input2)

# launch a session
sess = tf.Session()

# run graph on session and retrieve results
result = sess.run(output)

# close the session
sess.close()
```

- graph-related lines

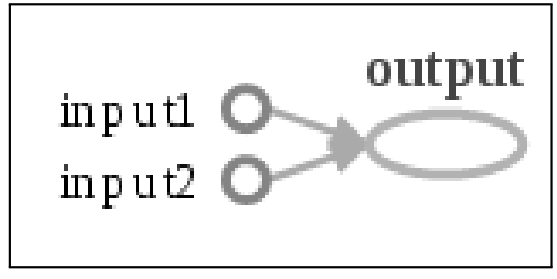


TensorFlow: basic usage

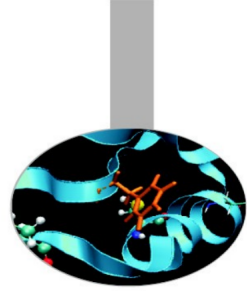


```
import tensorflow as tf

# define graph nodes (on default graph)
input1 = tf.constant([1.0, 1.0, 1.0, 1.0])
input2 = tf.constant([2.0, 2.0, 2.0, 2.0])
output = tf.add(input1, input2)

# launch a session
sess = tf.Session()

# run graph on session and retrieve results
result = sess.run(output)

# close the session
sess.close()
```

- graph-related lines
- session-related lines

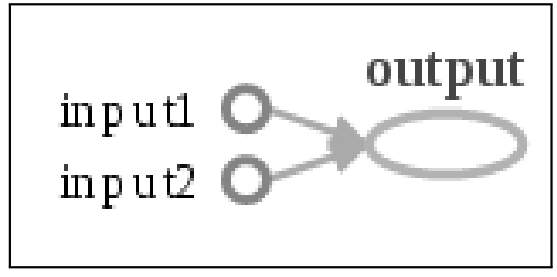


TensorFlow: basic usage

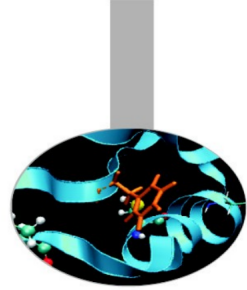


```
import tensorflow as tf

# define graph nodes (on default graph)
input1 = tf.constant([1.0, 1.0, 1.0, 1.0])
input2 = tf.constant([2.0, 2.0, 2.0, 2.0])
output = tf.add(input1, input2)

# launch a session
sess = tf.Session()

# run graph on session and retrieve results
result = sess.run(output)

# close the session
sess.close()
```

- graph-related lines
- session-related lines
- evaluation-related lines



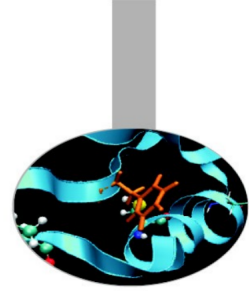
TensorFlow Graph and Session classes



An instance of tf.Graph class:

- represents, as a directed graph, the dataflow of the computation we want to perform;
- graph nodes are called operations (instances of tf.Operation class);
  - not necessarily a mathematical operation, but also a variable/constant definition, ...
- each operation involves zero or more instances of tf.Tensor class, it produces zero or more instances of same class;
- a tf.Tensor is a multi-dimensional array;
- in these examples we are using the default graph (accessible through tf.get_default_graph() function).

TensorFlow Graph and Session classes



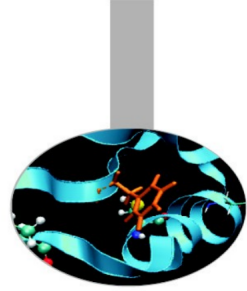
An instance of tf.Graph class:

- represents, as a directed graph, the dataflow of the computation we want to perform;
- graph nodes are called operations (instances of tf.Operation class);
  - not necessarily a mathematical operation, but also a variable/constant definition, ...
- each operation involves zero or more instances of tf.Tensor class, it produces zero or more instances of same class;
- a tf.Tensor is a multi-dimensional array;
- in these examples we are using the default graph (accessible through tf.get_default_graph() function).

An instance of tf.Session class:

- is used for running graph nodes;
- may own resources, so use Session.close() method when you are done with it;
- it can be customizable
  - you chan choose hardware where evaluation is performed;
- it can provide results of node evaluations through Session.run()) method.

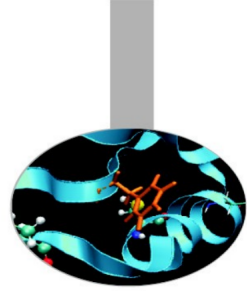
Tensors: variables, constants, placeholders



An instance of tf.Variable class:

- implements the mathematical concept of variable;
- maintains it's value accross different Session.run() calls;
- can be assigned a new value through the function tf.assign() (and others of type tf.assign...())
- it requires an initialization step.

Tensors: variables, constants, placeholders



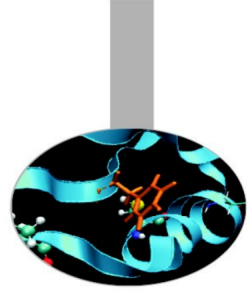
An instance of tf.Variable class:

- implements the mathematical concept of variable;
- maintains it's value accross different Session.run() calls;
- can be assigned a new value through the function tf.assign() (and others of type tf.assign...())
- it requires an initialization step.

Other Tensors:

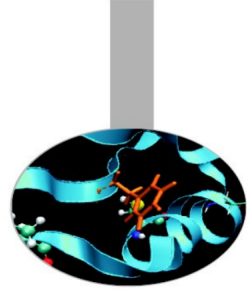
- use tf.constant() function to initialize constant tensors;
- use tf.placeholder() for a tensor that will be fed later during the evaluation.

approx. minimization of a univariate functional (I)



```
import tensorflow as tf
import numpy as np

# define variable and placeholders
a_op = tf.placeholder( tf.float32, shape=(1) )
b_op = tf.placeholder( tf.float32, shape=(1) )
x_op = tf.Variable( [ 2.0 ], dtype=tf.float32 )

y_op = a_op * x_op**2  + b_op * x_op

optimizer = tf.train.AdamOptimizer(0.1)
min_step = optimizer.minimize( y_op )

init_op = tf.global_variables_initializer()
```

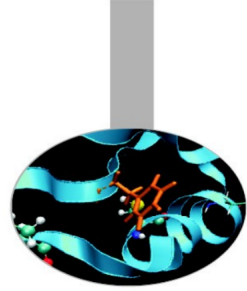
approx. minimization of a univariate functional (I)



```
import tensorflow as tf
import numpy as np

# define variable and placeholders
a_op = tf.placeholder( tf.float32, shape=(1) )
b_op = tf.placeholder( tf.float32, shape=(1) )
x_op = tf.Variable( [ 2.0 ], dtype=tf.float32 )

y_op = a_op * x_op**2  + b_op * x_op

optimizer = tf.train.AdamOptimizer(0.1)
min_step = optimizer.minimize( y_op )

init_op = tf.global_variables_initializer()
```

Define tensors: placeholders for *a* and *b* parameters, variable for *x*.

approx. minimization of a univariate functional (I)

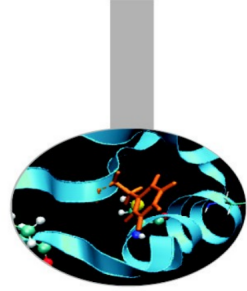


```
import tensorflow as tf
import numpy as np

# define variable and placeholders
a_op = tf.placeholder( tf.float32, shape=(1) )
b_op = tf.placeholder( tf.float32, shape=(1) )
x_op = tf.Variable( [ 2.0 ], dtype=tf.float32 )

y_op = a_op * x_op**2  + b_op * x_op

optimizer = tf.train.AdamOptimizer(0.1)
min_step = optimizer.minimize( y_op )

init_op = tf.global_variables_initializer()
```

Overloaded operators will call pointwise math operations: what we are actually calling is:

```
y_op = tf.add( tf.multiply( tf.pow( x_op, 2 ), a_op),
        tf.multiply(x_op, b_op ) )
```

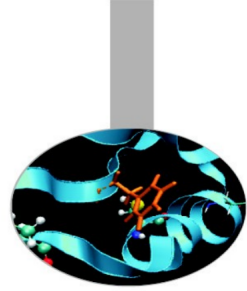
approx. minimization of a univariate functional (I)



```
import tensorflow as tf
import numpy as np

# define variable and placeholders
a_op = tf.placeholder( tf.float32, shape=(1) )
b_op = tf.placeholder( tf.float32, shape=(1) )
x_op = tf.Variable( [ 2.0 ], dtype=tf.float32 )

y_op = a_op * x_op**2  + b_op * x_op

optimizer = tf.train.AdamOptimizer(0.1)
min_step = optimizer.minimize( y_op )

init_op = tf.global_variables_initializer()
```

Define optimization method, pass the function to be optimized.

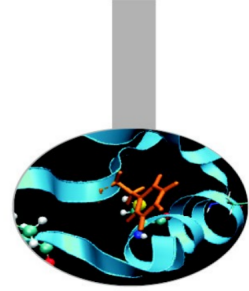
approx. minimization of a univariate functional (I)



```
import tensorflow as tf
import numpy as np

# define variable and placeholders
a_op = tf.placeholder( tf.float32, shape=(1) )
b_op = tf.placeholder( tf.float32, shape=(1) )
x_op = tf.Variable( [ 2.0 ], dtype=tf.float32 )

y_op = a_op * x_op**2  + b_op * x_op

optimizer = tf.train.AdamOptimizer(0.1)
min_step = optimizer.minimize( y_op )

init_op = tf.global_variables_initializer()
```

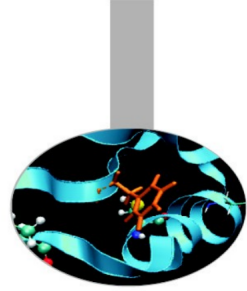
Define an operation for variables initialization.

approx. minimization of a univariate functional (II)



```
# launch a session
sess = tf.Session()

# initialize variables
sess.run(init_op)

a = [ 1.0]
b = [-2.0]
y = []
for ii in range( 100 ):
    _,y_p = sess.run([min_step, y_op],
                     feed_dict = { a_op: a, b_op: b } )
    y.append( y_p )

x = sess.run(x_op)

sess.close()
```

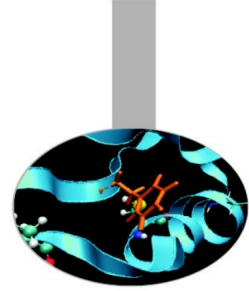
approx. minimization of a univariate functional (II)



```
# launch a session
sess = tf.Session()

# initialize variables
sess.run(init_op)

a = [ 1.0]
b = [-2.0]
y = []
for ii in range( 100 ):
    _,y_p = sess.run([min_step, y_op],
                     feed_dict = { a_op: a, b_op: b } )
    y.append( y_p )

x = sess.run(x_op)

sess.close()
```

Define a session and initialize all variables of the graph.

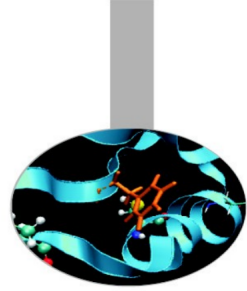
approx. minimization of a univariate functional (II)



```
# launch a session
sess = tf.Session()

# initialize variables
sess.run(init_op)

a = [ 1.0]
b = [-2.0]
y = []
for ii in range( 100 ):
    _,y_p = sess.run([min_step, y_op],
                     feed_dict = { a_op: a, b_op: b } )
    y.append( y_p )

x = sess.run(x_op)

sess.close()
```

Define some Python variables: function parameters *a* and *b* and a vector *y* to hold function value at each iteration.

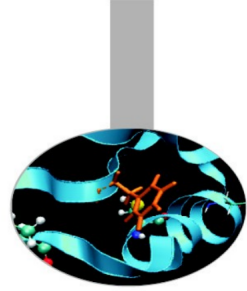
approx. minimization of a univariate functional (II)



```
# launch a session
sess = tf.Session()

# initialize variables
sess.run(init_op)

a = [ 1.0]
b = [-2.0]
y = []
for ii in range( 100 ):
    _,y_p = sess.run([min_step, y_op],
                     feed_dict = { a_op: a, b_op: b } )
    y.append( y_p )

x = sess.run(x_op)

sess.close()
```

Implement main loop: note the Python dictionary feed_dict and the return values of sess.run() method.

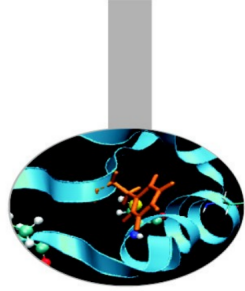
approx. minimization of a univariate functional (II)



```
# launch a session
sess = tf.Session()

# initialize variables
sess.run(init_op)

a = [ 1.0]
b = [-2.0]
y = []
for ii in range( 100 ):
    _,y_p = sess.run([min_step, y_op],
                     feed_dict = { a_op: a, b_op: b } )
    y.append( y_p )

x = sess.run(x_op)

sess.close()
```

Retrieve the estimated value of the minimum point.

approx. minimization of a univariate functional (III)



```
import matplotlib.pyplot as plt

print('solution: %f, (true: %f)'
  % (x, - np.divide(b, np.multiply( [a],  2.0) )))

plt.plot(y)
plt.xlabel('iterations')
plt.ylabel('function value')
plt.show()
```

Estimated value is 1.002937, real is 1.0.